目录

[1. Two Sum 1](#_Toc489801459)

[2. Add Two Numbers 2](#_Toc489801460)

[3. Longest Substring Without Repeating Characters 3](#_Toc489801461)

[4. Median of Two Sorted Arrays 4](#_Toc489801462)

[5. Longest Palindromic Substring 5](#_Toc489801463)

# 1. Two Sum

Given an array of integers, return **indices** of the two numbers such that they add up to a specific target.

You may assume that each input would have ***exactly*** one solution, and you may not use the *same* element twice.

**Example:**

Given nums = [2, 7, 11, 15], target = 9,

Because nums[**0**] + nums[**1**] = 2 + 7 = 9,

return [**0**, **1**].

Solution1 Brute Force O(n2)/O(1)

**public** **int[]** **twoSum(int[]** nums**,** **int** target**)** **{**

**for** **(int** i **=** 0**;** i **<** nums**.**length**;** i**++)** **{**

**for** **(int** j **=** i **+** 1**;** j **<** nums**.**length**;** j**++)** **{**

**if** **(**nums**[**j**]** **==** target **-** nums**[**i**])** **{**

**return** **new** **int[]** **{** i**,** j **};**

**}**

**}**

**}**

**throw** **new** IllegalArgumentException**(**"No two sum solution"**);**

**}**

Solution2 Two-pass Hash Table O(n)/O(n)

**public** **int[]** **twoSum(int[]** nums**,** **int** target**)** **{**

Map**<**Integer**,** Integer**>** map **=** **new** HashMap**<>();**

**for** **(int** i **=** 0**;** i **<** nums**.**length**;** i**++)** **{**

map**.**put**(**nums**[**i**],** i**);**

**}**

**for** **(int** i **=** 0**;** i **<** nums**.**length**;** i**++)** **{**

**int** complement **=** target **-** nums**[**i**];**

**if** **(**map**.**containsKey**(**complement**)** **&&** map**.**get**(**complement**)** **!=** i**)** **{**

**return** **new** **int[]** **{** i**,** map**.**get**(**complement**)** **};**

**}**

**}**

**throw** **new** IllegalArgumentException**(**"No two sum solution"**);**

**}**

Solution3 One-pass Hash Table O(n)/O(n)

**public** **int[]** **twoSum(int[]** nums**,** **int** target**)** **{**

Map**<**Integer**,** Integer**>** map **=** **new** HashMap**<>();**

**for** **(int** i **=** 0**;** i **<** nums**.**length**;** i**++)** **{**

**int** complement **=** target **-** nums**[**i**];**

**if** **(**map**.**containsKey**(**complement**))** **{**

**return** **new** **int[]** **{** map**.**get**(**complement**),** i **};**

**}**

map**.**put**(**nums**[**i**],** i**);**

**}**

**throw** **new** IllegalArgumentException**(**"No two sum solution"**);**

**}**

# 2. Add Two Numbers

You are given two **non-empty** linked lists representing two non-negative integers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

**Input:** (2 -> 4 -> 3) + (5 -> 6 -> 4)  
**Output:** 7 -> 0 -> 8

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

Solution1 O(max(m,n))/O(max(m,n))

**public** ListNode **addTwoNumbers(**ListNode l1**,** ListNode l2**)** **{**

ListNode dummyHead **=** **new** ListNode**(**0**);**

ListNode p **=** l1**,** q **=** l2**,** curr **=** dummyHead**;**

**int** carry **=** 0**;**

**while** **(**p **!=** **null** **||** q **!=** **null)** **{**

**int** x **=** **(**p **!=** **null)** **?** p**.**val **:** 0**;**

**int** y **=** **(**q **!=** **null)** **?** q**.**val **:** 0**;**

**int** sum **=** carry **+** x **+** y**;**

carry **=** sum **/** 10**;**

curr**.**next **=** **new** ListNode**(**sum **%** 10**);**

curr **=** curr**.**next**;**

**if** **(**p **!=** **null)** p **=** p**.**next**;**

**if** **(**q **!=** **null)** q **=** q**.**next**;**

**}**

**if** **(**carry **>** 0**)** **{**

curr**.**next **=** **new** ListNode**(**carry**);**

**}**

**return** dummyHead**.**next**;**

**}**

# 3. Longest Substring Without Repeating Characters

Given a string, find the length of the **longest substring** without repeating characters.

**Examples:**

Given "abcabcbb", the answer is "abc", which the length is 3.

Given "bbbbb", the answer is "b", with the length of 1.

Given "pwwkew", the answer is "wke", with the length of 3. Note that the answer must be a **substring**, "pwke" is a *subsequence* and not a substring.

Solution1 Brute Force O(n3)/O(min(n,m))

**public** **class** **Solution** **{**

**public** **int** **lengthOfLongestSubstring(**String s**)** **{**

**int** n **=** s**.**length**();**

**int** ans **=** 0**;**

**for** **(int** i **=** 0**;** i **<** n**;** i**++)**

**for** **(int** j **=** i **+** 1**;** j **<=** n**;** j**++)**

**if** **(**allUnique**(**s**,** i**,** j**))** ans **=** Math**.**max**(**ans**,** j **-** i**);**

**return** ans**;**

**}**

**public** **boolean** **allUnique(**String s**,** **int** start**,** **int** end**)** **{**

Set**<**Character**>** set **=** **new** HashSet**<>();**

**for** **(int** i **=** start**;** i **<** end**;** i**++)** **{**

Character ch **=** s**.**charAt**(**i**);**

**if** **(**set**.**contains**(**ch**))** **return** **false;**

set**.**add**(**ch**);**

**}**

**return** **true;**

**}**

**}**

Solution2 Sliding Window O(2n)=O(n)/O(min(m,n))

**public** **class** **Solution** **{**

**public** **int** **lengthOfLongestSubstring(**String s**)** **{**

**int** n **=** s**.**length**();**

Set**<**Character**>** set **=** **new** HashSet**<>();**

**int** ans **=** 0**,** i **=** 0**,** j **=** 0**;**

**while** **(**i **<** n **&&** j **<** n**)** **{**

*// try to extend the range [i, j]*

**if** **(!**set**.**contains**(**s**.**charAt**(**j**))){**

set**.**add**(**s**.**charAt**(**j**++));**

ans **=** Math**.**max**(**ans**,** j **-** i**);**

**}**

**else** **{**

set**.**remove**(**s**.**charAt**(**i**++));**

**}**

**}**

**return** ans**;**

**}**

**}**

Solution3 Sliding Window Optimized HasMap O(n)/O(min(m,n))

**public** **class** **Solution** **{**

**public** **int** **lengthOfLongestSubstring(**String s**)** **{**

**int** n **=** s**.**length**(),** ans **=** 0**;**

Map**<**Character**,** Integer**>** map **=** **new** HashMap**<>();** *// current index of character*

*// try to extend the range [i, j]*

**for** **(int** j **=** 0**,** i **=** 0**;** j **<** n**;** j**++)** **{**

**if** **(**map**.**containsKey**(**s**.**charAt**(**j**)))** **{**

i **=** Math**.**max**(**map**.**get**(**s**.**charAt**(**j**)),** i**);**

**}**

ans **=** Math**.**max**(**ans**,** j **-** i **+** 1**);**

map**.**put**(**s**.**charAt**(**j**),** j **+** 1**);**

**}**

**return** ans**;**

**}**

**}**

Solution4 Sliding Window Optimized Charset Table O(n)/O(m)

**public** **class** **Solution** **{**

**public** **int** **lengthOfLongestSubstring(**String s**)** **{**

**int** n **=** s**.**length**(),** ans **=** 0**;**

**int[]** index **=** **new** **int[**128**];** *// current index of character*

*// try to extend the range [i, j]*

**for** **(int** j **=** 0**,** i **=** 0**;** j **<** n**;** j**++)** **{**

i **=** Math**.**max**(**index**[**s**.**charAt**(**j**)],** i**);**

ans **=** Math**.**max**(**ans**,** j **-** i **+** 1**);**

index**[**s**.**charAt**(**j**)]** **=** j **+** 1**;**

**}**

**return** ans**;**

**}**

**}**

# 4. Median of Two Sorted Arrays

There are two sorted arrays nums1 and nums2 of size m and n respectively.

Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

Example 1:

nums1 = [1, 3]

nums2 = [2]

The median is 2.0

Example 2:

nums1 = [1, 2]

nums2 = [3, 4]

The median is (2 + 3)/2 = 2.5

Solution O(log(min(n1,n2)))

**①**template <typename T>

T trivialMedian ( Vector<T>& S1, int lo1, int n1, Vector<T>& S2, int lo2, int n2 ) {

int hi1 = lo1 + n1, hi2 = lo2 + n2;

Vector<T> S;

while ( ( lo1 < hi1 ) && ( lo2 < hi2 ) ) {

while ( ( lo1 < hi1 ) && S1[lo1] <= S2[lo2] ) S.insert ( S1[lo1 ++] );

while ( ( lo2 < hi2 ) && S2[lo2] <= S1[lo1] ) S.insert ( S2[lo2 ++] );

}

while ( lo1 < hi1 ) S.insert ( S1[lo1 ++] );

while ( lo2 < hi2 ) S.insert ( S1[lo2 ++] );

((n1 + n2) % 2 == 0) ?

return (S[(n1 + n2) / 2] + S[(n1 + n2) / 2 - 1]) / 2 : return S[( n1 + n2 ) / 2];

}

**②**template <typename T>

T median ( Vector<T>& S1, int lo1, int n1, Vector<T>& S2, int lo2, int n2 ) {

if ( n1 > n2 ) return median ( S2, lo2, n2, S1, lo1, n1 );

if ( n2 < 6 )

return trivialMedian ( S1, lo1, n1, S2, lo2, n2 );

if ( 2 \* n1 < n2 )

return median ( S1, lo1, n1, S2, lo2 + ( n2 - n1 - 1 ) / 2, n1 + 2 - ( n2 - n1 ) % 2 );

int mi1 = lo1 + n1 / 2;

int mi2a = lo2 + ( n1 - 1 ) / 2;

int mi2b = lo2 + n2 - 1 - n1 / 2;

if ( S1[mi1] > S2[mi2b] )

return median ( S1, lo1, n1 / 2 + 1, S2, mi2a, n2 - ( n1 - 1 ) / 2 );

else if ( S1[mi1] < S2[mi2a] )

return median ( S1, mi1, ( n1 + 1 ) / 2, S2, lo2, n2 - n1 / 2 );

else //S1保留，S2左右同时缩短

return median ( S1, lo1, n1, S2, mi2a, n2 - ( n1 - 1 ) / 2 \* 2 );

}

# 5. Longest Palindromic Substring

Given a string s, find the longest palindromic substring in s. You may assume that the maximum length of s is 1000.

Example:

Input: "babad"

Output: "bab"

Note: "aba" is also a valid answer.

Example:

Input: "cbbd"

Output: "bb"

Solution1 Expand Around Center O(n2)/O(1)

public String longestPalindrome(String s) {

int start = 0, end = 0;

for (int i = 0; i < s.length(); i++) {

int len1 = expandAroundCenter(s, i, i);

int len2 = expandAroundCenter(s, i, i + 1);

int len = Math.max(len1, len2);

if (len > end - start) {

start = i - (len - 1) / 2;

end = i + len / 2;

}

}

return s.substring(start, end + 1);

}

private int expandAroundCenter(String s, int left, int right) {

int L = left, R = right;

while (L >= 0 && R < s.length() && s.charAt(L) == s.charAt(R)) {

L--;

R++;

}

return R - L - 1;

}

Solution2 Manacher’s Algorithm

# 11. Container With Most Water

Given *n* non-negative integers *a1*, *a2*, ..., *an*, where each represents a point at coordinate (*i*, *ai*). *n* vertical lines are drawn such that the two endpoints of line *i* is at (*i*, *ai*) and (*i*, 0). Find two lines, which together with x-axis forms a container, such that the container contains the most water.

Note: You may not slant the container and *n* is at least 2.

Solution Two Pointer Approach O(n)/O(1)

# 99. Recover Binary Search Tree

Two elements of a binary search tree (BST) are swapped by mistake.

Recover the tree without changing its structure.

**Note:**  
A solution using O(*n*) space is pretty straight forward. Could you devise a constant space solution?

# 100. Same Tree

Given two binary trees, write a function to check if they are the same or not.

Two binary trees are considered the same if they are structurally identical and the nodes have the same value.

**Example 1:**

**Input:** 1 1

/ \ / \

2 3 2 3

[1,2,3], [1,2,3]

**Output:** true

**Example 2:**

**Input:** 1 1

/ \

2 2

[1,2], [1,null,2]

**Output:** false

**Example 3:**

**Input:** 1 1

/ \ / \

2 1 1 2

[1,2,1], [1,1,2]

**Output:** false

**Solution1：**

**public** boolean isSameTree(TreeNode p, TreeNode q) {

**if**(p == null && q == null) **return** true;

**if**(p == null ^ q == null) **return** false;

**if**(p.**val** == q.**val**)

**return** isSameTree(p.left, q.left) && isSameTree(p.right, q.right);

**return** false; }

**Solution2：**

**public** **boolean** isSameTree(TreeNode p, TreeNode q) {

Stack<TreeNode> stack\_p = **new** Stack <> ();

Stack<TreeNode> stack\_q = **new** Stack <> ();

**if** (p != **null**) stack\_p.**push**( p ) ;

**if** (q != **null**) stack\_q.**push**( q ) ;

**while** (!stack\_p.isEmpty() && !stack\_q.isEmpty()) {

TreeNode pn = stack\_p.**pop**() ;

TreeNode qn = stack\_q.**pop**() ;

**if** (pn.val != qn.val) **return** **false** ;

**if** (pn.right != **null**) stack\_p.**push**(pn.right) ;

**if** (qn.right != **null**) stack\_q.**push**(qn.right) ;

**if** (stack\_p.**size**() != stack\_q.**size**()) **return** **false** ;

**if** (pn.left != **null**) stack\_p.**push**(pn.left) ;

**if** (qn.left != **null**) stack\_q.**push**(qn.left) ;

**if** (stack\_p.**size**() != stack\_q.**size**()) **return** **false** ; }

**return** stack\_p.**size**() == stack\_q.**size**() ; }

11.14.66.70.20.26.55.88.89.17.22.46.79.42.53.21

104.191.258.226.461.122.238.202.136.108.253.231

155.206.283.141.198.172.160.349.153.110.

148.328.74.96.134.22.79

123.42.138.33.240.173.179

316.150.139

188.283.238

136.217

206.141.328.160.142

121.122.134.316

349.374.278.153.50.287.33.74.240

123.198.338

104.226.100.235.110.108

136.137.371.191.338.268.231